



C# Avançado

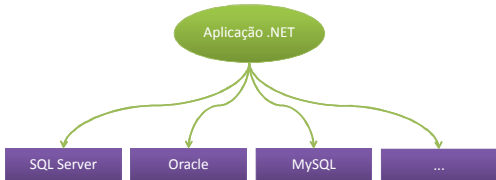
Bancos de Dados com ADO.NET

Softblue
cursos online

Tópicos Abordados

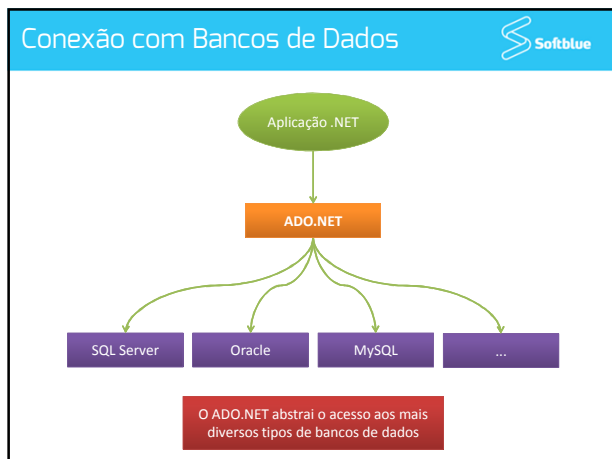
- ADO.NET na conexão com bancos de dados
- ADO.NET Data Providers
- Trabalhando com conexões
 - Externalizando dados do provider
- Executando comandos
- Comandos parametrizados
- Transações

Conexão com Bancos de Dados

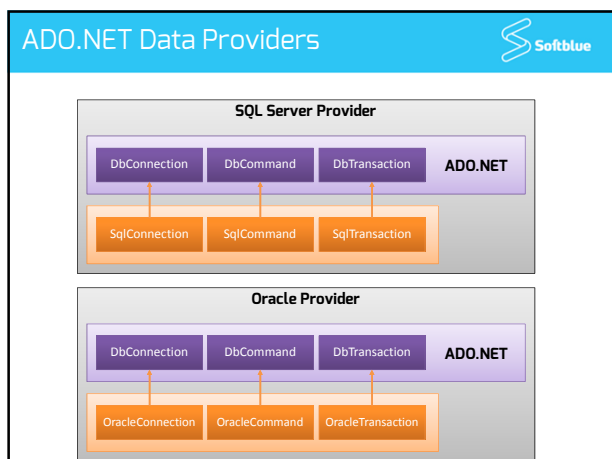


```
graph TD; A([Aplicação .NET]) --> B[SQL Server]; A --> C[Oracle]; A --> D[MySQL]; A --> E[...];
```

Cada SGBD (RDBMS) se comporta de forma diferente



- ### ADO.NET Data Providers
- Um data provider é responsável por permitir a comunicação com determinado SGBD
 - Cada SGBD tem um provider específico
 - A plataforma .NET já traz alguns providers nativos
 - Microsoft SQL Server
 - ODBC
 - OLE DB
 - Os providers estão disponíveis no assembly *System.Data.dll*



Provider Factory

- Um provider factory permite criar objetos de um provider específico

```
DbProviderFactory factory =
    DbProviderFactories.GetFactory("System.Data.SqlClient");
```

Nome do provider

```
DbConnection conn = factory.CreateConnection();
DbCommand cmd = factory.CreateCommand();
```

Código independente de provider

```
SqlConnection conn = new SqlConnection();
SqlCommand cmd = new SqlCommand();
```

Código específico do SQL Server

Conexão com o Banco de Dados

- Obter um objeto *DbConnection*

```
using (DbConnection conn = factory.CreateConnection())
{
    conn.ConnectionString = @"Data Source=(local)\SQLEXPRESS;
    Initial Catalog=testedb;Integrated Security=True";
    conn.Open();
}
```

Bloco using

```
DbConnection conn = factory.CreateConnection();
try
{
    conn.Open();
}
finally
{
    conn.Close();
}
```

Bloco try..finally

Connection Strings

- Cada provider define um formato e informações para a connection string
- A classe *ConnectionStringBuilder* pode ser usada para facilitar a criação da connection string

```
SqlConnectionStringBuilder sb = new SqlConnectionStringBuilder();
sb.DataSource = @"(local)\SQLEXPRESS";
sb.InitialCatalog = "testedb";
sb.IntegratedSecurity = true;

conn.ConnectionString = sb.ConnectionString;
```

Externalizando Dados do Provider



- Deixar informações sobre o provider diretamente no código não é uma boa prática
- O nome do provider e a connection string podem ir para um arquivo de configuração
 - *App.config*

```
<configuration>
  <appSettings>
    <add key="provider" value="System.Data.SqlClient" />
  </appSettings>
  <connectionStrings>
    <add name="db" connectionString="Data Source=(local)\SQLEXPRESS;
      Initial Catalog=testedb;Integrated Security=True" />
  </connectionStrings>
</configuration>
```

Externalizando Dados do Provider



- A classe *ConfigurationManager* é utilizada para ler as informações
 - Assembly: *System.Configuration.dll*
 - Namespace: *System.Configuration*

```
String provider = ConfigurationManager.AppSettings["provider"];
String cn = ConfigurationManager.ConnectionStrings["db"].ConnectionString;
```

Criando Comandos



- Depois de estabelecida a conexão, o próximo passo é a criação de comandos, que serão executados no banco de dados
 - INSERT, DELETE, UPDATE, SELECT
- Um objeto do tipo *DbCommand* é utilizado

```
DbCommand cmd = factory.CreateCommand();
cmd.Connection = conn;
cmd.CommandText = "SELECT nome FROM contato";
```

A conexão e a query devem ser associadas ao comando

Executando Comandos



- A execução de um comando é feita através dos seguinte métodos

Método	Tipo de Comando	Retorno
<code>ExecuteNonQuery()</code>	INSERT, UPDATE, DELETE	<code>int</code>
<code>ExecuteReader()</code>	SELECT	<code>DbDataReader</code>
<code>ExecuteScalar()</code>	SELECT	<code>object</code>

Executando Comandos



```
...  
cmd.CommandText = "INSERT INTO contato(nome) VALUES ('Maria')";  
int num = cmd.ExecuteNonQuery();
```

Número de registros afetados

```
...  
cmd.CommandText = "SELECT nome, idade, FROM contato";  
using (DbDataReader result = cmd.ExecuteReader())  
{  
    while (result.Read())  
    {  
        string nome = (string)result["nome"];  
        int idade = (int)result["idade"];  
    }  
}
```

Loop enquanto houver registros

Extrai os dados do data reader

É possível também usar índices com base na posição da coluna

Executando Comandos



```
...  
cmd.CommandText = "SELECT MAX(idade) FROM contato";  
object obj = cmd.ExecuteScalar();  
if (!Convert.IsDBNull(obj))  
{  
    int max = (int)obj;  
}
```

Retorna o valor da primeira coluna do primeiro registro

Verifica se o valor é nulo antes de fazer o casting

- `DBNull` é uma classe que representa um valor nulo vindo do banco de dados

```
if (count == DBNull.Value)  
{  
    ...  
}
```

Outra forma de checar a nulidade

Comandos Parametrizados



- Exemplo com concatenação

```
void Inserir(string nome, int idade, DbConnection conn, DbProviderFactory f)
{
    using (DbCommand cmd = f.CreateCommand())
    {
        cmd.Connection = conn;
        cmd.CommandText = "INSERT INTO contato(nome, idade)
        VALUES('" + nome + "', " + idade + ")";
        cmd.ExecuteNonQuery();
    }
}
```

- A query é criada e enviada toda vez para o banco de dados, a fim de ser executada
- A solução para este problema é criar um comando parametrizado

Comandos Parametrizados



- Exemplo de comando parametrizado

```
void Inserir(string nome, int idade, DbConnection conn, DbProviderFactory f)
{
    using (DbCommand cmd = factory.CreateCommand())
    {
        cmd.CommandText = "INSERT INTO contato(nome, idade) VALUES(@Nome, @Idade)";
        cmd.Connection = conn;

        DbParameter param = f.CreateParameter();
        param.ParameterName = "@Nome";
        param.Value = nome;
        cmd.Parameters.Add(param);

        param = f.CreateParameter();
        param.ParameterName = "@Idade";
        param.Value = idade;
        cmd.Parameters.Add(param);

        cmd.ExecuteNonQuery();
    }
}
```

Placeholders

Parâmetros para substituir os placeholders

O parse da query ocorre apenas uma vez, o que acelera a sua execução

Comandos Parametrizados



- O .NET converte o tipo de dado do C# em um tipo de dado entendido pelo banco de dados
- É possível controlar a conversão via programação

```
param.ParameterName = "@Nome";
param.Value = nome;
param.DbType = DbType.String;
```

Conversão válida para qualquer provider

```
param.ParameterName = "@Nome";
param.Value = nome;
param.DbType = SqlDbType.NVarChar;
```

Conversão válida apenas para o SQL Server

- A documentação do ADO.NET detalha como funcionam as conversões para os mais diversos tipos de dados existentes

Transações



- Uma transação é uma operação atômica
 - Ou ela executa por completo, ou não executa
 - Não existe a possibilidade de ela executar apenas parcialmente
- O exemplo mais clássico de transação é uma transferência bancária de um valor
 - Duas operações
 - Saque na conta de origem
 - Depósito na conta de destino
 - Ambas precisam executar de forma atômica

Iniciando uma Transação



- No ADO.NET, uma transação é iniciada através do método *BeginTransaction()*, da classe *DbConnection*

```
DbTransaction transaction = conn.BeginTransaction();
```

Objeto que representa a transação

- A transação termina com um commit ou um rollback

```
transaction.Commit();
```

Efetiva a transação

```
transaction.Rollback();
```

Desfaz a transação